REMARKS

Claims 1-20 remain pending in the present application as amended. Independent claims 1, 8, and 14 have been amended. No claims have been added or canceled. Applicants respectfully submit that no new matter has been added to the application by the Amendment. In particular, Applicants respectfully submit that the added language in the claims regarding the pen service, the pen input component, and the stylus input subsystem among other things is disclosed in the published application at least at paragraphs [0057], [0063], and [0066]-[0071] and in connection with Fig. 4.

Telephone Conversation With Examiner

Examiner Joseph is thanked for the telephone conversation conducted on September 29, 2009. Proposed amendments were discussed. Asserted art was discussed. Differences between Figure 3 and Figure 4 of the instant application were discussed. No agreements were reached.

Claim Rejections

The Examiner has now rejected the claims under 35 U.S.C. § 103 as being obvious over admitted prior art of the present application as disclosed in connection with Fig. 3, and further in view of Price et al. (U.S. Patent Pub. No. 2002/0120607) and Vermeire et al. (U.S. Patent Pub. No. 2001/0025372). Applicants respectfully traverse the Section 103 rejection insofar as it may be applied to the claims as amended. In particular, Applicants respectfully submit that the admitted prior art does not show an application with a stylus input subsystem as a managed component and a pen input component as an unmanaged component, or that the pen input component interacts with a separate pen service written in unmanaged code, particularly in the manner now recited in independent claims 1, 8, and 14 as amended. Also, Applicants respectfully submit that the admitted prior art does not show a retrieval command being submitted by such a stylus input subsystem by way of an exposed P-Invoke-style interface to such a pen input component, or that the pen input component exposes the P-Invoke-style interface, as is now recited in claims 1, 8, and 14.

Preliminarily, Applicants wish to point out that the claims have been amended after a close review of the application as filed, particularly regarding Fig. 4 and the subject matter disclosed therein. In doing so, Applicants note that a fair amount of confusion has been created regarding the principle elements recited in the independent claims and the functionality thereof. Accordingly, Applicants after such close review have amended the independent claims to address such confusion. In particular, the claims as amended now recite an application with a stylus input subsystem as a managed component and a pen input component as an unmanaged component[1], and also a pen service separate from the recited application and written in unmanaged code.

In the present application, and as was previously pointed out, pen data from a digitizer of a computing device or information derived from such pen data is employed by an application on the computing device, where the application includes managed code. The pen data relates to movement of a stylus with respect to the digitizer, and thus includes information relating to one or more locations of the stylus with respect to the digitizer. Notably, although the application employs managed code to process the pen data, the computing device employs a component (such as a pen service) to initially receive the incoming pen data, where the component is not managed code but instead is written as unmanaged code.

As previously discussed, managed code and unmanaged code relate to use of a framework on the computing device such as the .NET framework supplied by MICROSOFT Corporation of Redmond, Washington. Such .NET framework in particular includes a large library of pre-coded solutions to common programming problems, a runtime or virtual machine that manages the execution of programs written specifically for the framework, and a set of tools for configuring and building applications. The pre-coded solutions that form the framework's Base Class Library cover a large range of programming needs in a number of areas, including user interface, data access, database connectivity, cryptography, web application development,

[1] Applicants note here that the application at issue is referred to in the Specification and Drawings as being a 'Managed Application', which is somewhat unfortunate if not confusing inasmuch as the application includes both managed and unmanaged components, as is plainly shown in Fig. 4, e.g.

numeric algorithms, and network communications. The class library is used by programmers who combine it with their own code to produce applications.

Programs written for the .NET Framework (i.e., 'managed code') execute in a software environment that manages the program's runtime requirements. Also part of the .NET Framework, this runtime environment is known as the Common Language Runtime (CLR). The CLR provides the appearance of an application virtual machine so that programmers need not consider the capabilities of the specific processor that will execute the program. As a result, managed code is executed under the CLR in a manner independent of the processor of the computing device. In contrast, 'unmanaged code' is not written for such framework and is not executed under the CLR. Instead, such unmanaged code is native to and executed directly by the processor of the computing device.

As was previously pointed out, interaction between managed and unmanaged code is commonly required, such as for example to transfer pen data from a digitizer to a .NET application. Accordingly, a framework such as the .NET Framework provides a mechanism to access functionality that is implemented in programs that execute outside the .NET environment. For example, and as set forth in the application at paragraph 0009, a callable run-time wrapper as shown in Fig. 3 may be invoked to retrieve information from a native component, where every call from managed space to the native component passes through standard interop data marshalling (data transfer) as established by the runtime callable wrapper. However, and significantly, such standard interop data marshalling (data transfer) as employed in the .NET framework, for example, extends across a metaphorical border between managed and unmanaged code and therefore is relatively slow, which can be an impediment to high data throughput. Moreover, the wrapper is external to an application and therefore represents an additional layer through which data must pass prior to reaching the application.

Accordingly, in the present application and as shown in Fig. 4, data transfer is instead achieved by way of shared memory and pointers as well as by employing entities that do not extend across the aforementioned metaphorical border between managed and unmanaged code, as is shown in Fig. 4. By doing so, packet transfer speed for pen data information may be

increased by a factor of 2-5 times as compared with the standard CLR interop scheme (paragraph [0055]).

As now recited in claim 1, the present innovation may be embodied as a process for transferring pen data between unmanaged and managed code on a computing device. The unmanaged code is code native to and executed directly by a processor of the computing device, and the managed code is code executed in a common language run-time environment of a framework operating on the computing device. Thus, the common language run-time environment of the framework executes the managed code independent of a type of the processor of the computing device.

In the process of claim 1, as amended, pen data is received in a *pen service* on the computing device, where the pen service is written in unmanaged code and is separate from the application. The pen data is generated by a digitizer of the computing device upon movement of a stylus with respect to a surface of the digitizer, and the pen data includes at least one location on the digitizer of the stylus. Information related to the pen data is transferred by the pen service to a mutual exclusion shared memory on the computing device designated to be non-simultaneously shared between unmanaged code and managed code, and the pen service also transfers a pointer that points to the information in the shared memory to a loaded pen input component.

As now recited, the pen input component is loaded into and is therefore a part of the application on the computing device, and is written in unmanaged code and is designated to communicate with the pen service for the application. As also now recited, the pen input component further transfers the received pointer to a stylus input subsystem of the application on the computing device, where the stylus input subsystem is separate from the pen input component and is written in managed code and executed in the common language run-time environment. The unmanaged code of the pen input component of the application transfers the received pointer to the managed code of the stylus input subsystem of the application by way of a P-Invoke-style interface exposed therebetween. As further now recited, a retrieval command is

submitted by the stylus input subsystem by way of an exposed P-Invoke-style interface to the pen input component, and the pen input component exposes the P-Invoke-style interface.

The stylus input subsystem receives the transferred pointer from the pen input component, and submits the received pointer with a retrieval command to the shared memory to retrieve the information from the shared memory by way of the transferred pointer. That is, the stylus input subsystem as recited in claim 1 does not communicate directly with the pen service to obtain the information therefrom, in the manner of that which is shown in Fig. 3. Instead, the pen service and the stylus input subsystem of claim 1 employ the shared memory and the pointer thereto as an intermediary drop-box, in the manner of Fig. 4, with the result being that the aforementioned direct communication is not required. As may be appreciated, such direct communication of Fig. 3 is slower, and therefore less desirable, because the direct communication takes place across the managed-unmanaged boundary. In contrast, the indirect communication of Fig. 4 is faster, and therefore more desirable, because the transfers are limited to a pointer and simple memory calls.

Independent claims 8 and 14 as amended each recite subject matter akin to the process of claim 1 as amended, albeit in the form of a system (claim 8) and a computer-readable medium (claim 14).

As amended, independent claims 1, 8, and 14 now recite an application with a stylus input subsystem and a pen input component, and a pen service separate from the application. Applicants respectfully submit that such elements are clearly shown in Fig. 4, and are also clearly different from that shown in connection with Fig. 3. In particular, Fig. 3 employs components 305, 306 *separate from* the application 307 to perform the transfer of pen data across the aforementioned managed-unmanaged boundary. Based thereon, Applicants respectfully submit that the actions performed by Fig. 3 are substantially different from the actions now recited in claims 1, 8, and 14. Additionally, Fig. 3 does not employ a P-Invoke-style interface to effectuate such transfer across such boundary, as is now recited in claims 1, 8, and 14. Moreover, and as was pointed out, Fig. 3 is inferior to the subject matter recited in claims 1,

8, and 14 particularly inasmuch as throughput is increased by a factor of 2-5 times as compared with Fig. 3.

As amended, independent claims 1, 8, and 14 also now recite a retrieval command being submitted by the stylus input subsystem by way of the exposed P-Invoke-style interface to the pen input component, and that the pen input component exposes the P-Invoke-style interface. Applicants respectfully submit that Fig. 3 and the accompanying discussion also does not disclose or even suggest such specific P-Invoke-related features.

The Price reference is cited only for the purpose of disclosing the use of a shared memory, and the Vermeire reference is cited only for the purpose of disclosing the use of pointers. Accordingly, Applicants respectfully submit that the Price and Vermeire references are otherwise inapposite the subject matter now recited in claims 1, 8, and 14. Thus, such references also do not disclose or even suggest the application with a stylus input subsystem and a pen input component, and a pen service separate from the application, as are now recited in claims 1, 8, and 14; the actions now recited in claims 1, 8, and 14; and the use of the P-Invoke-style interface to effectuate transfers, as is now recited in claims 1, 8, and 14.

Thus, Applicants respectfully submit that the admitted prior art in combination with the cited Price and Vermeire references fails to disclose or even suggest all of the above-mentioned elements as are now recited in independent claims 1, 8, and 14 of the present application as amended. Therefore, Applicants respectfully submit that such admitted prior art and such references cannot be combined to make obvious the subject matter recited in the independent claims as amended or the claims depending therefrom. Accordingly, Applicants respectfully request reconsideration and withdrawal of the Section 103 rejections.

CONCLUSION

In view of the foregoing Amendment and Remarks, Applicants respectfully submit that the present application including claims 1-20 is in condition for allowance and such action is respectfully requested.

Respectfully Submitted,

Date: November 17, 2009

/Joseph F. Oriti/
Joseph F. Oriti
Registration No. 47,835

Woodcock Washburn LLP
Cira Centre
2929 Arch Street, 12th Floor
Philadelphia, PA 19104-2891
Telephone: (215) 568-3100
Facsimile: (215) 568-3439